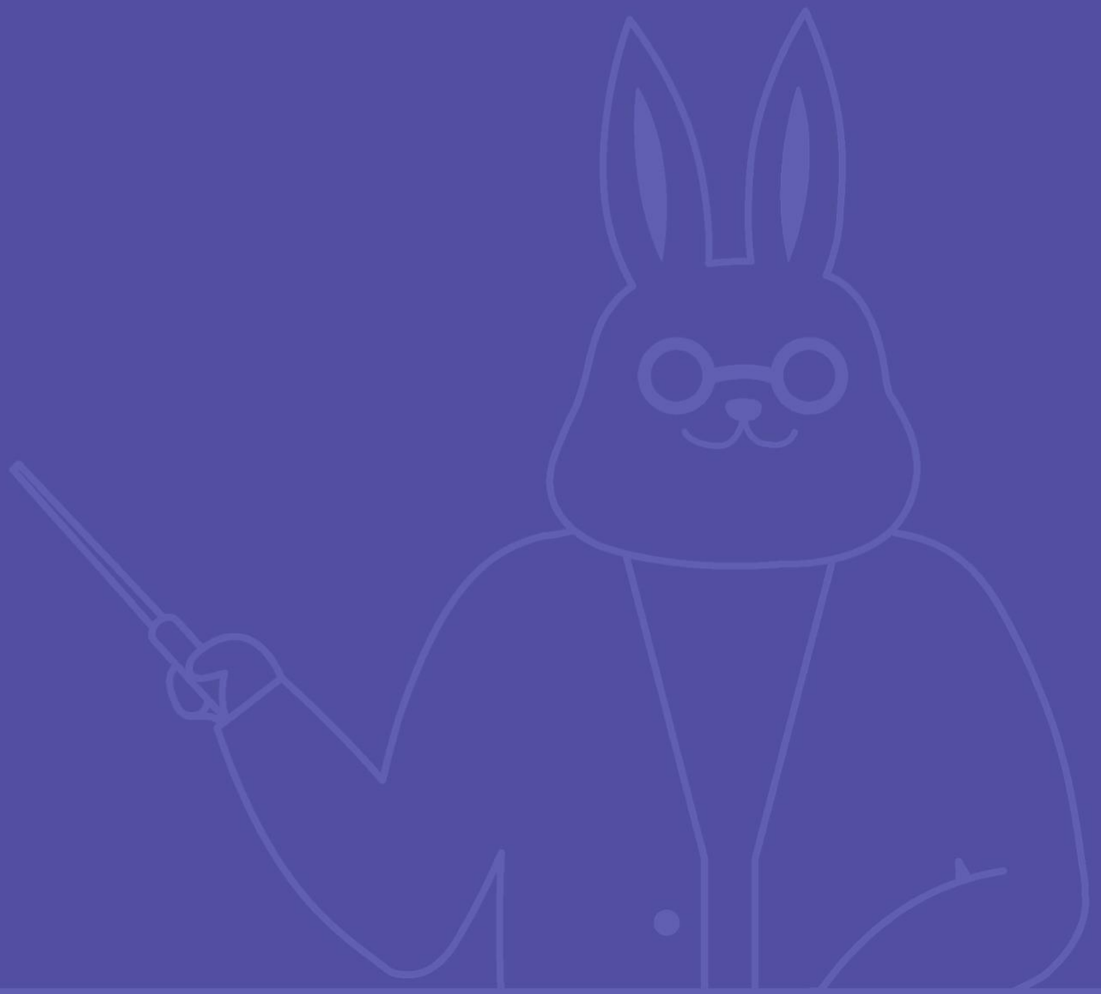
Flask 기초

02 서비스 구현하기



/* elice */

목차

🎯 수강목표

Flask의 기본 기능을 이해한다.

Blueprint와 Jinja Template를 사용해 Flask 서버를 다룰 수 있습니다.

Data를 다룰 수 있다.

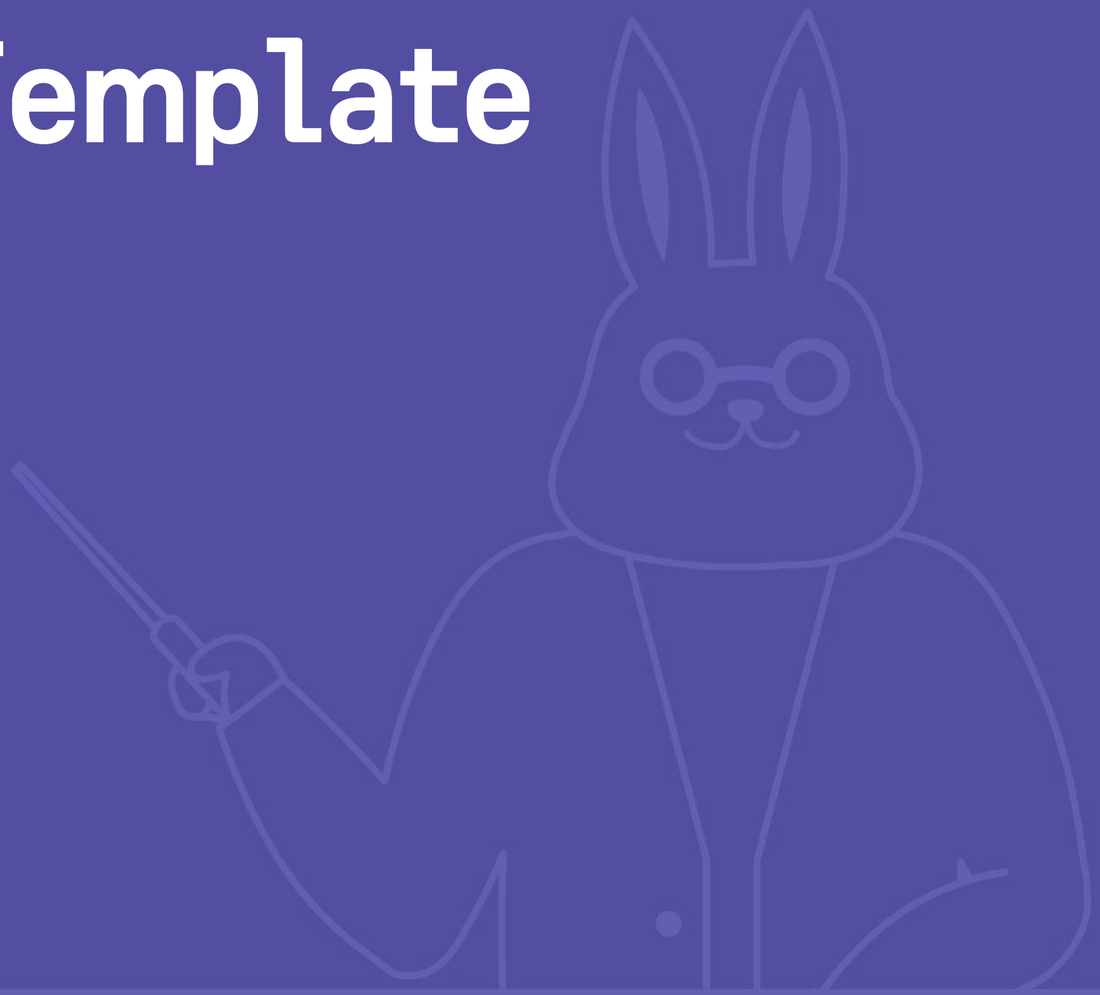
CRUD의 개념을 알고, Flask 서버에서 CRUD의 기능을 작성할 수 있습니다.

세션, 쿠키, 로깅에 대해서 이해한다.

세션과 쿠키를 이해할 수 있고, 로그인에 적용할 수 있습니다. 또한, 서버의 로깅에 대해 이해하고 사용할 수 있습니다.

01

Blueprint와 Jinja Template



Blueprint란?

Flask의 기능이 점점 늘어날수록, 자연스럽게 코드의 양이 증가합니다.

이때, **Blueprint**를 사용해서 길어진 코드를 모듈화해주어
수정 개발과 유지보수에 용이하게 코드를 관리할 수 있습니다.

Blueprint를 사용하지 않았을 때

flask

```
from flask import Flask,jsonify
app = Flask(__name__)

@app.route("/", methods=['GET'])
def home_route():
    return jsonfiy('home')

@app.route("/first",methods=['GET'])
def first_route():
    return jsonify('first page')
```

```
@app.route("/second",methods=['GET'])
def second_route():
    return jsonfiy('second page')

@app.route("/third",methods=['GET'])
def third_route():
    return jsonfiy('third page')

#...생략
if __name__ == '__main__':
    app.run(debug=True)
```

Blueprint를 사용했을 때

app.py

```
from flask import Flask
from first_api import bp

app = Flask(__name__)
app.register_blueprint(bp)

if __name__ == '__main__':
    app.run(debug=True)
```

first_api.py

```
from flask import Blueprint,jsonify
bp = Blueprint('bp',__name__)

@bp.route('/first',methods=['GET'])
def first_route():
    return jsonify('first page')

@bp.route('/second',methods=['GET'])
def second_route():
    return jsonify('second page')
```

Jinja2란?

Jinja2는 Python에서 가장 많이 사용되는 **템플릿**입니다.

서버에서 받아온 **데이터를 효과적으로 보여주고**
비교적 **간략한 표현으로 데이터를 가공**할 수 있습니다.

Jinja2 Template에서 데이터 넘겨주기 – 단일변수

app.py

```
@app.route("/")
def elice():
    return render_template(
                'index.html' ,
                data = 'elice'
            )
```

index.html

```
<html>
    <head>
        <title> jinja example </title>
    </head>
    <body>
        {{ data }}
    </body>
</html>
```

Jinja2 Template에서 데이터 넘겨주기 – list

app.py

```
@app.route("/")
def elice():
    my_list = [1,2,3,4,5]
     return render_template(
           'index.html' ,
           data = my_list
           )
```

index.html

```
<html>
    <head>
        <title> jinja example </title>
    </head>
    <body>
        {{ data }}
        {% for d in data %}
            {{ d }}
        {% endfor %}
    </body>
</html>
```

Jinja2 Template에서 데이터 넘겨주기 – dictionary

app.py

```python
@app.route("/")
def elice():
    my_data = {'name': 'elice'}
     return render_template(
           'index.html' ,
           data = my_data
           )
```

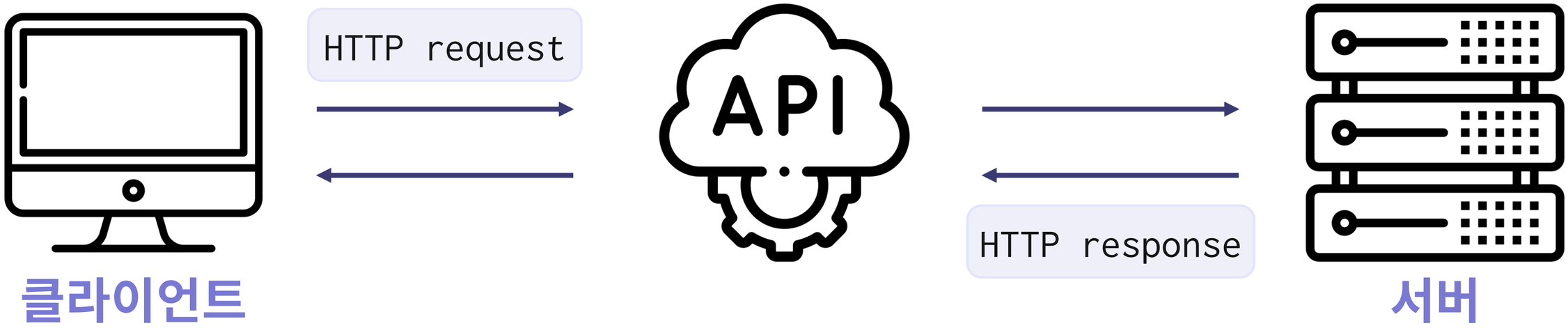
index.html

```html
<html>
    <head>
        <title> jinja example </title>
    </head>
    <body>
        {{ data.get('name') }}
    </body>
</html>
```

02

게시판을 위한 CRUD 설계 및 제작

API 동작 원리



CRUD란?

Create → 데이터의 생성

Read → 데이터 조회

Update → 데이터 수정

Delete → 데이터 삭제

API 설계

게시판 만들기

게시판을 만들기 위한 **최소한의 기능만을 구현**해봅니다.

CRUD

게시판 내용 **생성**

게시판 **조회**

게시판 **수정 및 추가**

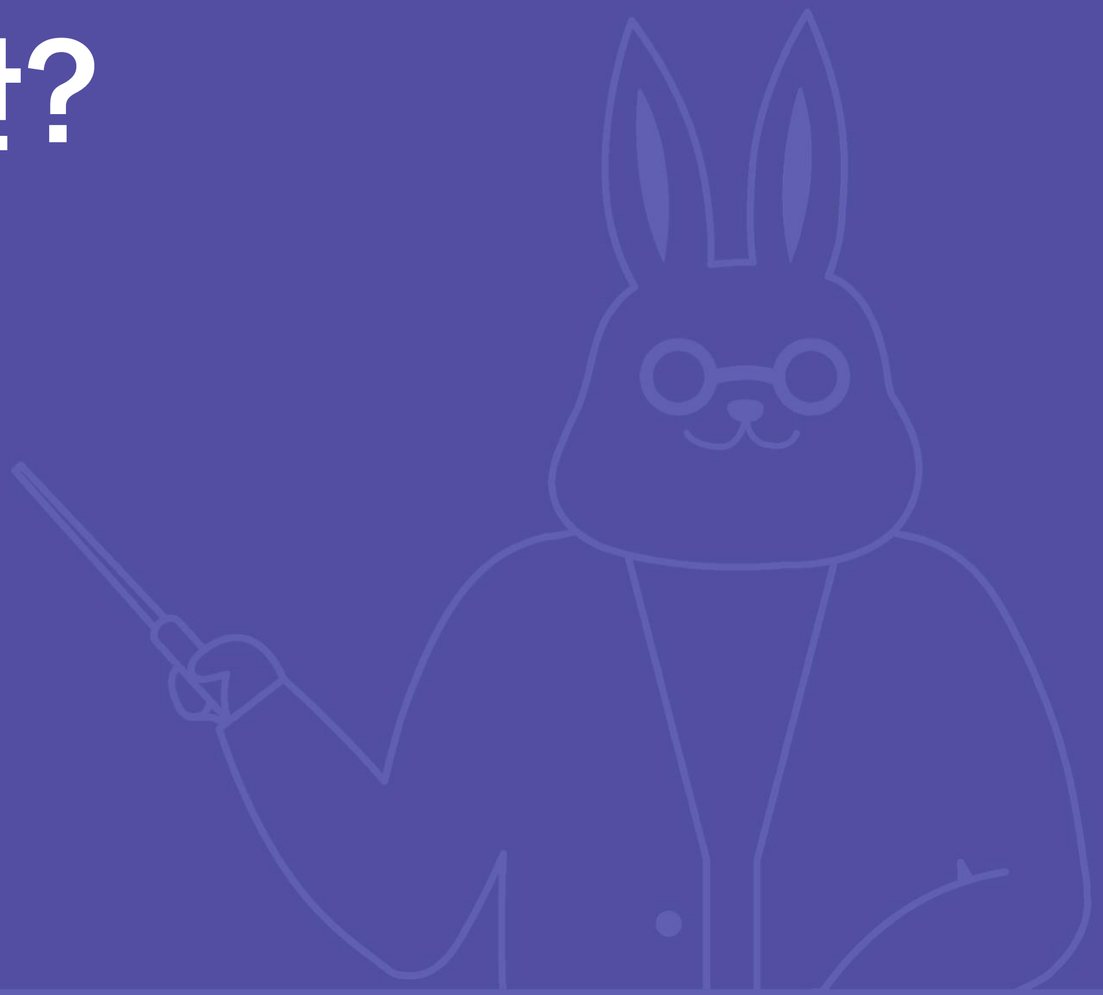
게시판 내용 **삭제**

CRUD

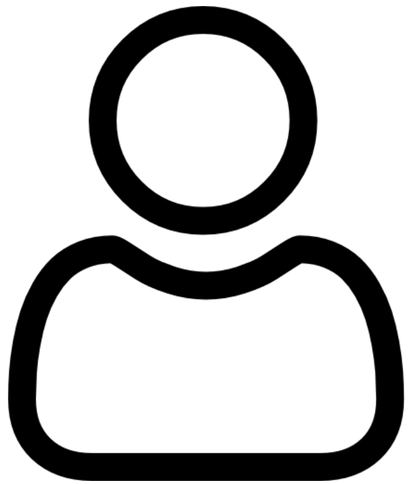
CRUD	HTTP Method	DB 명령어
Create	POST	INSERT
Read	GET	SELECT
Update	PUT, PATCH	UPDATE
Delete	DELETE	DELETE

03

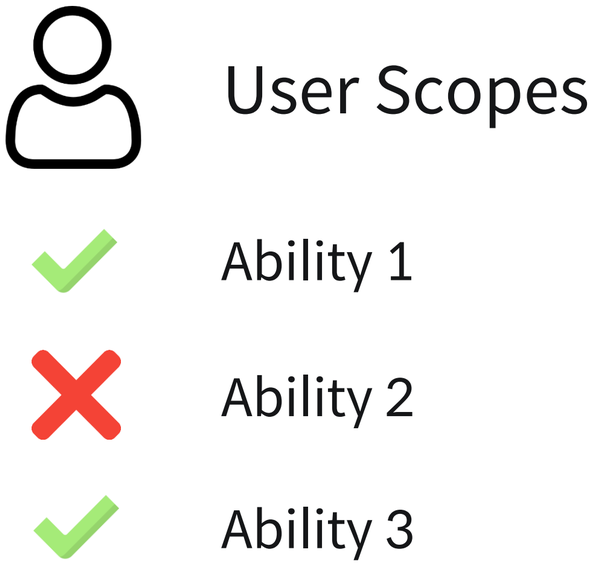
Authentication이란?



Authentication vs Authorization



WHO ARE YOU?



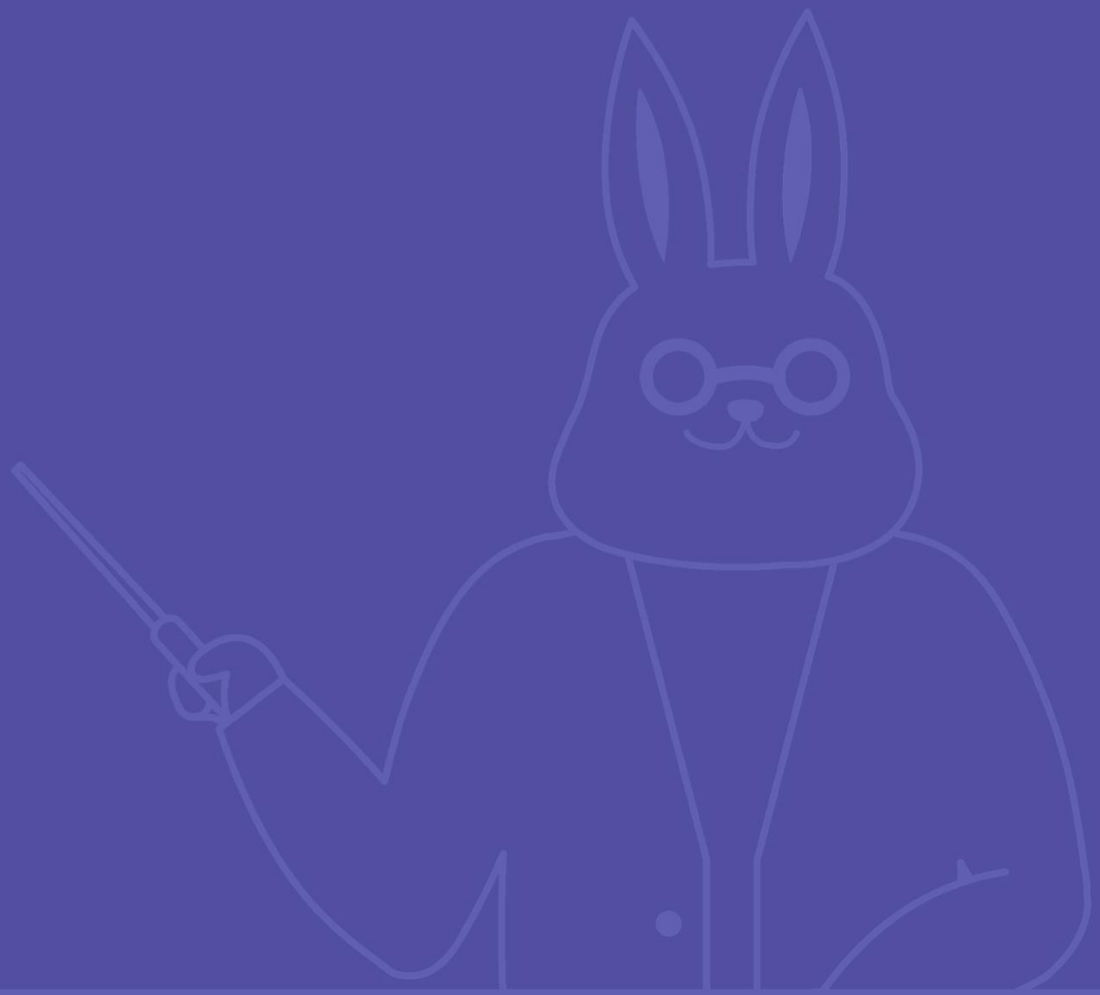
WHAT YOU CAN DO?

Authentication

Authentication이란 사용자가 누구인지 확인하는 절차를 말합니다.
→ 회원가입하고 로그인하는 과정

04

로그인 기능 구현



쿠키

클라이언트에 저장되는 키/값 이 들어 있는 데이터

사용자가 따로 요청하지 않아도, Request 시에 자동으로 서버에 전송합니다.

세션

쿠키를 기반으로 하지만 서버 측에서 관리하는 데이터

클라이언트에 고유 ID를 부여하고 클라이언트에 알맞은 서비스를 제공합니다.
서버에서 관리하기 때문에 보안이 쿠키보다 우수합니다.

로그인 예제

example

```
user_id = request.form['user_id']
user_pw = request.form['user_pw']
user ={'user_id': 'elice', 'user_pw': '1234'}
if user is not None:
    if user_id == user['user_id'] and user_pw == user['user_pw']:
        session['login'] = user.id
        return jsonify({"result":"success"})
    else:
        return jsonify({"result":"fail"})
```

- request로 받아온 로그인 정보 (user_id, user_pw)를 변수에 저장합니다.
- 데이터베이스에 user_id와 같은 데이터가 있는지 찾아옵니다.
- 입력된 user_pw와 저장된 패스워드가 같은지 체크합니다.

로그아웃 예제

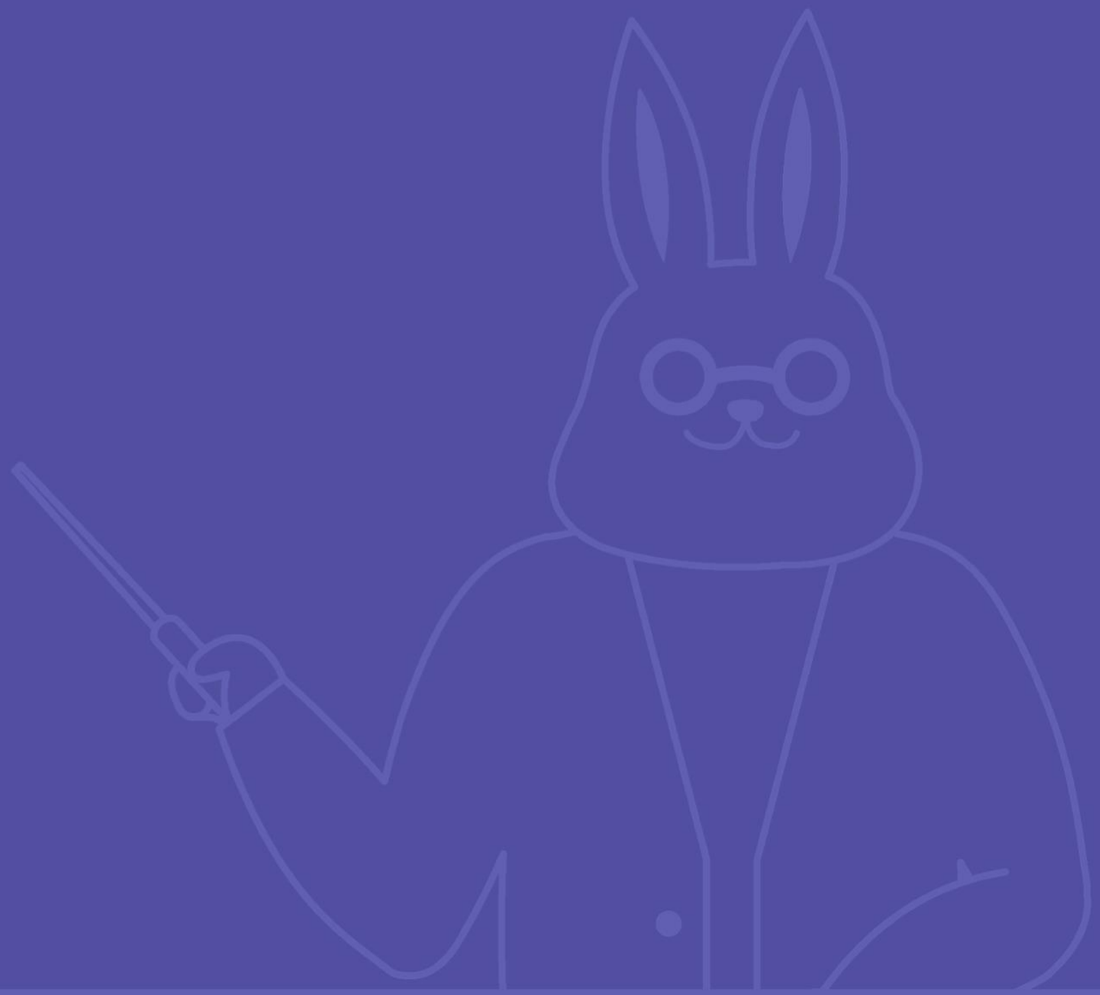
example

```
session['login'] = None
```

- 로그아웃을 할 때는 정보를 받을 필요가 없습니다.
- 현재 저장된 session의 데이터를 비워 주면 기능이 완료됩니다.

05

로깅



로깅이란?

로깅은 **프로그램이 작동할 때 발생하는 이벤트를 추적하는 행위**를 말합니다.

프로그램의 문제들을 파악하고 유지보수 하는 데 사용되며, 로깅을 통해 발생한 에러를 추적할 수 있습니다.

로깅 - level

DEBUG < INFO < WARNING < ERROR < CRITICAL

기본 로거 레벨 세팅은 **WARNING**이기 때문에 설정 없이 **INFO**, **DEBUG**를 출력할 수 없습니다.

DEBUG: 상세한 정보
INFO: 일반적인 정보
WARNING: 예상치 못하거나 가까운 미래에 발생할 문제
ERROR: 에러 로그. 심각한 문제
CRITICAL: 프로그램 자체가 실행되지 않을 수 있는 문제

로깅 - level

DEBUG < INFO < WARNING < ERROR < CRITICAL

기본 로거 레벨 세팅은 **WARNING**이기 때문에 설정 없이 **INFO**, **DEBUG**를 출력할 수 없습니다.

example

```
import logging

if __name__ == '__main__':

        logger.info("hello elice!")
```

example

```
import logging

if __name__ == '__main__':

        logger = logging.getLogger()
        logger.setlevel(logging.DEBUG)
        logger.info("hello elice!")
```

Flask에서 로깅

에러가 발생했을 때 Flask의 **logger**를 사용하여 에러를 확인 가능합니다.

Flask에서 로깅

Flask에서 기본적으로 logging을 제공합니다.

example

```
from flask import Flask
app = Flask(__name__)

if __name__ == '__main__':
        app.logger.info("test")
        app.logger.debug("debug test")
        app.logger.error("error test")
        app.run()
```

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
이바울, 이재성

강사
이바울

감수자
이바울

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io